Handout
Project

Algoritmos e Estruturas de Dados
NOVA FCT
Bernardo Toninho

October 10, 2024 (Updated November 21)

Contents

Changelog

Updates and modifications to the statement will be reported here.

21 November: Clarified definition of train overtaking and details pertaining to Task 2 functionality.

29 October: Changed print of line name in best schedule command for print of train number. Added clarification on the output of the command.

17 October: Added clarification on processing commands (Section 5); the non-existence of repeated stations when inserting a line (Section 5.3.1); the valid assumptions when inserting a new schedule and the definition of invalid schedule(Section 5.3.5); the best schedule command (Section 5.3.9).

10 October: Initial Publication

1 Introduction

This document describes the project of the Algorithms and Data Structures course of the 2nd year of the Degree in Computer Science (Eng. Informática). Students can and should ask any questions regarding the projects to the lecturers.

Project in groups of 2 students to be carried out in accordance with the principles of the Informatics Department's Code of Ethics. No individual groups without the explicit authorization of a professor.

Project with 2 phases of delivery. See submission details, submission dates and possible versions in Section 6.1.

2 Railway Network Support System

In this section you will find information about the problem, the associated concepts, the syntax used and the general specification of the system. In section 5.3, each of the features will be described in the form of commands that must be made available.

3 Objective

The aim of this work is to develop a system to support the management of a railway network. A railway network is composed of a set of rail lines, each containing a sequence of associated stations.

In general terms, the aim is to support the creation and removal of lines; creating and removing train timetables; consulting stations and timetables of a given line; and searching the lines of a given station.

4 Concepts

Each line on our rail network contains a sequence of stations. For example, the *Cascais* line contains the following sequence of stations: Cais do Sodré, Santos, Alcântara, Belém, Algés, Cruz Quebrada, Caxias, Paço d'Arcos, Santo Amaro, Oeiras, Carcavelos, Parede, São Pedro, São João, Estoril, Monte Estoril and Cascais. The Cais do Sodré and Cascais stations are terminal stations.

Each line is associated with a set of daily timetables. It is assumed that the timetables are the same for every day of the week. Each timetable defines the time at which a train arrives (and leaves) from each of the stations on the line, on a given route. Therefore, each timetable is defined by a train number, and a schedule comprised of a sequence of pairs (station, time). The direction of travel is defined by the departure station, which is a terminal station. For example, a timetable on the Cascais line timetable, made by train 19423 departing from the Cais do Sodré station (direction Cais do Sodré - Cascais), is as follows:

Nº Comboio: 19423
Serviços a bordo:
Observações:
Ⓐ

Estações	
Cais do Sodré	9:45
Santos	-
Alcântara	9:49
Belém	-
Algés	9:54
Cruz Quebrada	-
Caxias	-
Paço de Arcos	-
Santo Amaro	-
Oeiras	10:02
Carcavelos	-
Parede	-
S. Pedro	10:08
S. João	10:11
Estoril	10:13
Monte Estoril	10:15
Cascais	10:17

Notes:

- On a given route, a train does not necessarily stop at every stations on the line;
- There is no overtaking between trains, i.e. the order of passing the trains at the stations is the same as the order in which they they left;
- A station can belong to several lines simultaneously.

5 System specification

The application should work as follows: after starting, the application will read commands from the standard input (System.in), processing them one by one and sending the responses to the standard output (System.out); termination will occur when the the appropriate command is given in the standard input.

Data persistence must be ensured between consecutive executions. This means that, before terminating execution, the system must save its state to disk, using Java's serialization features. In the next execution of the program, the stored data must be loaded from disk and reconstituted before processing any command.

Note: For each command, all its input must be processed before reporting success or failure.

5.1 Syntax

The application's interface is intended to be very simple, so that it can be used in different environments and and, in the case of the output, to automate the testing process. For these reasons input and output must comply with the strict format indicated in the next section. You can assume that the input is syntactically correct, but you should not assume that it is semantically correct.

5.2 Data types

Line and station names are sequences of characters. The train number is an integer. Times are represented in hh:mm format, where hh is a sequence of two digits denoting an integer between 0 and 23 (inclusive) and integer between 0 and 23 (inclusive) and mm is a two-digit sequence denoting an integer between 0 and 59 (inclusive).

The number of lines, stations and timetables is not limited above. We could, however, envisage a number of 500 lines, 1000 stations and 10000 timetables. It is expected there will be a proportional number of insertions and deletions in the system, but that the number of listing operations will be higher.

Note: For the first phase, assume that there are only a few dozen lines, stations and timetables.

The system should not be case-sensitive when handling names and commands.

5.3 Syntax of operations to be implemented

In this section we give a detailed description of the syntax of each of the operations that the application must implement. The symbol ↲ represents a line change. Error conditions are listed in the order they should be validated. For instance, in the schedule insert operation (Section 5.3.5), the message "Linha inexistente." will be shown (and only this message) if the line in question does not exist, regardless of whether or not the schedule is valid.

5.3.1 Line insertion

Input	`IL line-name`↲ `name-station-1`↲ ... `name-station-n`↲ ↲
Output (successful)	Inserção de linha com sucesso.↲
Output (unsuccessful)	Linha existente.↲

Insertion of a line, given a name and a non-empty list of station names. The error situation occurs if a line with the same name already exists in the system. You may assume (i.e., without verification) that the names of stations of a line are all different amongst themselves.

5.3.2 Remove line

Input	`RL line-name`↲
Output (successful)	Remoção de linha com sucesso.↲
Output (unsuccessful)	Linha inexistente.↲

Removal of a line with the given name. All stations on the line that do not belong to another line will be deleted. The error situation applies when the line does not exist in the system.

5.3.3 Listing the stations of a line

Entry	`CL line-name`↲
Output (successful)	`name-line`↲ `name-station-1`↲ ... `name-station-n`↲
Output (unsuccessful)	Linha inexistente.↲

Lists the stations on a given line, if the line exists. The stations must be listed in the order given when inserting the line.

5.3.4 Listing the lines of a station

Input	`CE station-name`↲
Output (successful)	station-name↲ `name-line-1`↲ ... title-row-n↲
Exit (unsuccessful)	Estação inexistente.↲

Lists the lines of a given station, if it exists. The list in lexicographic order. **Note: This functionality will only be graded in Phase 2.**

5.3.5 Inserting a schedule

Entry	`IH line-name↲`
	`train-number↲`
	`name-station-1 hour-1↲`
	`...`
	`name-station-n hour-n↲`
	`↲`

Output (successful)	Criação de horário com sucesso.↲
Output (unsuccessful)	Linha inexistente.↲ Horário inválido.↲

When inserting a schedule, you may **assume** (i.e., without verification) that train numbers are unique. You may also **assume** that a given schedule never overflows onto the next day (i.e., all schedules begin and end on the same day). To insert a schedule, the first station indicated (station-name-1) must be one of the two terminal stations of the line in question and so the direction of travel. You can therefore define schedules for either of the two directions of a line. A schedule always has at least two stations. Note that a schedule need not include a stop at every station of the line.

A schedule is **invalid** when:

Phase 1: **its first station is not one of the two terminal stations of the line, its stations are not in the order specified in the line** and its times are not strictly increasing;

Phase 2: its times are not strictly increasing or **when overtaking (or simultaneity) is introduced in relation to another existing schedule for that direction of the line**. The definition of overtaking is given below.

Considering that a schedule is carried out by a specific train, it should not be be possible for two trains to depart at the same time from the same station, or to overtake each other at subsequent stations. In other words, considering the initial example of train 19423 departing from Cais do Sodré at 9:45 towards Cascais, we can see that it arrives at Alcântara at 9:49 and Oeiras at 10:02. Another train that departs from Cais do Sodré at 9:46, to arrive in Alcântara at 9:48 would have to overtake train 19423 and would therefore have an **invalid** schedule. Similarly, a train leaving Cais do Sodré at 9:40 and arriving at Algés at 9:55 would have to be overtaken by train 19423 (which arrives at Algés earlier, departing later from Cais do Sodré), thus making this schedule **invalid**. In other words, a schedule H that departs from a terminal station at time t must arrive at all its stops **before** any other schedule departing from the same terminal station at a time t' such that $t' > t$. Symmetrically, a schedule departing from a terminal station at time t must arrive at all its stops **later** than any other schedule departing from the same terminal station at a time t'' such that $t'' < t$. **Note:** For this definition you need only consider stations that are **shared** between schedules.

Note: in **Phase 1** you may assume that no input will contain schedules where trains depart simultaneously from the same station at the same time. You may also assume in **Phase 1** that no two schedules will contain trains that overtake each other.

5.3.6 Schedule removal

Entry	`RH line-name`↲ `station-name departure-time`↲
Output (successful)	Remoção de horário com sucesso. ↲
Output (unsuccessful)	Linha inexistente.↲ Horário inexistente.↲

Removal of a schedule, if the line and schedule exist.

5.3.7 Listing the schedules of a line

Entry	`CH line-name`↲ `departure-station-name`↲
Output (successful)	`train-number-1`↲ `station-name-11 time-11`↲ `...` `station-name-1j time-1j`↲ `...` `train-number-n`↲ `station-name-n1 time-n1`↲ `...` `station-name-ni time-ni`↲
Output (unsuccessful)	Linha inexistente.↲ Estação de partida inexistente.↲

Lists all the schedules for a given line, if the line exists and the departure station is a terminal station (determining the direction of travel). direction). The list should be sorted by departure time.

5.3.8 Trains by station

Input	`LC station-name`↲
Output (successful)	`Comboio train-number-1 time-1`↲ `...` `Comboio train-number-n time-n`↲
Output (unsuccessful)	Estação inexistente.↲

Lists all trains that pass by the given station in increasing order of departure time. The listed time should be that at which the train passes by the given station. When two trains pass by the station at the same time, the one with the lowest number should be listed first. **Note: this operation will only be evaluated in Phase 2.**

5.3.9 Best Timetable

Entry	`MH line-name`↲ `departure-station-name`↲ `destination-station-name` ↲ `expected-arrival-time` ↲
Output (successful)	`num-train`↲ `name-station-11 hour-11`↲ `...` `name-station-1j hour-1j`↲
Output (unsuccessful)	Linha inexistente.↲ Estação de partida inexistente.↲ Percurso impossível.↲

Determines, if possible, the "best" route between two stations (departure and destination) on a given line. In this context, the best route is the one that, **without changing trains**, arrives at the destination station as close to the expected arrival time as possible (but never later). If a route is found, all stops from the route should be printed (not just those between the departure and destination stations).

Note that the departure and destination stations may be any two stations of the line. If the destination station does not exist in the line, the schedule should be deemed impossible.

5.3.10 Terminate application

Input	`TA`↲
Output	Aplicação terminada.↲

6 Development

Students carrying out the project must be registered in some lab class. The project is carried out in groups of two students, preferably from the same lab class. Any deviations from this principle must be explicitly authorized by a teacher.

The work is implemented in Java 21, and there are two basic versions of the project:

- In the complete version I, evaluated between 0 and 20 values, it is not allowed to use the *java.util* package, with the exception of the *Scanner* class;
- In version J, rated between 0 and 8, you can use all interfaces and classes from the standard Java library can be used.

Students have the option of submitting a version with some interfaces and classes implemented according to version I and others according to version according to version J. In this case, the teachers will assign a grade depending on the number of resources to *java.util*, heavily penalizing these uses.

6.1 Deliveries and phasing

The work will be carried out incrementally, in two phases with already scheduled deadlines.

In the first phase, students will have to design their solution using the abstract data types and their respective data structures as presented by the end of lecture 5 (see Moodle class numbering), and may also use the TAD Dictionary if they feel it is appropriate (but not its implementation as a Hashtable). It is important to note that in the development of the work as a whole, it may be appropriate to implement variants of the data structures discussed in class, and students should discuss these variants with teachers.

The complete definition of valid schedule (Section 5.3.5), the lines by station listing command (Section 5.3.4) and the trains by station command (Section 5.3.8) will only be evaluated in Phase 2.

All operations must be implemented in the 2nd phase. The choice of abstract data types should be reviewed in the 2nd phase in order to improve the performance of the work from the 1st to the 2nd phase, taking into account the requirements of section 5.3. It is expected that some of the abstract data types used in the 1st phase solution are changed in the 2nd phase.

The program must be submitted in Mooshak (contests AED2425_Phase1 and AED2425_Phase2, respectively). In the last phase a final report on the project must also be submitted via Moodle. Each group must register for the Mooshak competitions. The name (Mooshak login) must be the concatenation of the numbers of the students in the group, separated by _ (the first number must be the smallest). For example, students nº3435 and nº3434 should have the Mooshak username 3434_3435. Projects will only be considered submitted if the Mooshak contest submission follows these rules.

The submission on Mooshak consists of a zip containing the source code code. The name and number of the students that make up the group must be inserted in the header of all the files, as follows:

```
/**
 * @author NAME1 (NUMBER2) email1
 * @author NAME2 (NUMBER2) email2
 */
```

6.2 Report

In the last phase, a final report of the project must be submitted on Moodle, containing the following:

- For each of the Abstract Data Types (Interfaces) defined in the assignment, briefly justify the chosen implementations in terms of data structures;
- For each of the operations described in Section 5.3, briefly describe their implementation in terms of the operations on the underlying data structures;
- The study of the temporal complexities of the operations described in Section 5.3, in the best case, worst case and expected case;
- The study of the spatial complexity of the proposed solution.

No templates or formats are defined for the report. All information to be included in the report is indicated here, in addition to the in addition to the reference to the version associated with the work (I, J or mixed).

7 Mooshak requirements

In order to submit the work to Mooshak, the source code must comply with the following rules:

- The complete source code must be saved in a ZIP file.
- The main class must be called Main and be located at the root of the archive (i.e. it must belong to the default package).
- The folders corresponding to the packages in the job must be located at the root of the file.
- The program can only create files in the current directory.

8 Evaluation

The assignment is compulsory for all students who do not have Frequência and gives Frequência. The assessment will cover all aspects: design, quality and efficiency of the solution, modularity, code structure and documentation, quality of the final report, etc. The 1st phase is worth 15% of the final grade. The 2nd phase is worth 25% of the final grade.

8.1 Functionality tests

The work must meet all the requirements specified in the Section 5.3. This check will be carried out automatically by the Mooshak system. At each stage:

- If the program does not pass any of the functionality tests, its authors will receive a score of 0 for the phase in question;
- If the program obtains the minimum number of points required for the evaluation of the phase, it will be subject to a preliminary evaluation that must be be confirmed at the end of the semester in a discussion or practical test.

8.2 Authorship check

At the end of the semester, all groups in a position to obtain Frequência will be subjected to a discussion or practical test to verify the authorship of the work. If a student misses the discussion of the work, they will not be awarded Frequência.

8.3 Important dates

- **1st phase:** October 30 – November 6 23:59.
- **2nd phase:** November 28 – December 5 23:59.
- **Discussions:** TBD.